Neural Networks (part 1)

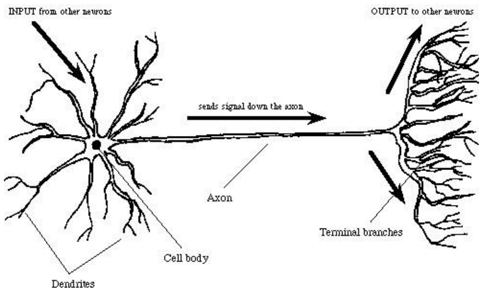
Goals for this lecture

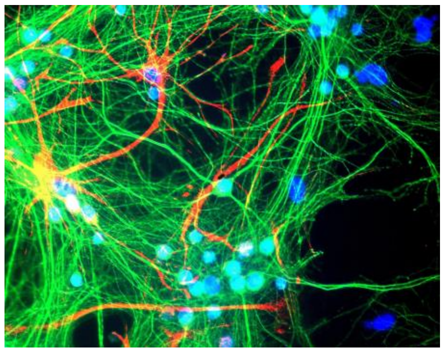
you should understand the following concepts

- perceptrons
- the perceptron training rule
- linear separability
- hidden units
- multilayer neural networks
- gradient descent
- stochastic (online) gradient descent
- activation functions
- sigmoid, hyperbolic tangent, ReLU
- objective (error, loss) functions
- squared error, cross entropy

Neural Networks

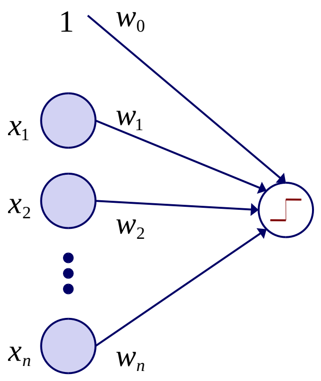
- a.k.a. *artificial neural networks*, *connectionist models*
- inspired by interconnected neurons in biological systems
- simple processing units
- each unit receives a number of real-valued inputs
- each unit produces a single real-valued output





Perceptrons

[McCulloch & Pitts, 1943; Rosenblatt, 1959; Widrow & Hoff, 1960]

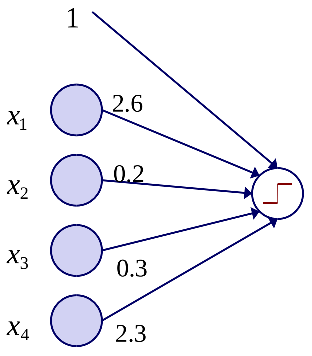


$$o = \begin{cases} 1 & \text{if } w_0 + \sum_{i=1}^{n} w_i x_i > 0 \\ 0 & \text{otherwise} \end{cases}$$

input units: represent given $\boldsymbol{x}$

output unit: represents binary classification

Perceptron example



$$o = \begin{cases} 1 & \text{if } w_0 + \sum_{i=1}^{n} w_i x_i > 0 \\ 0 & \text{otherwise} \end{cases}$$

features, class labels are represented numerically

$$\mathbf{x} = \langle 1,0,0,1 \rangle \quad w_0 + \sum_{i=1}^{n} w_i x_i = -0.1 \qquad o = 0$$

$$\langle \qquad \rangle \qquad n$$

$$\mathbf{x} = \langle 1,0,1,1 \rangle \quad w_0 + \sum_{i=1} w_i x_i = 0.2 \qquad o = 1$$

Training a perceptron

1. randomly initialize weights

2. iterate through training instances until convergence

2a. calculate the output for the given instance

$$o = \begin{cases} 1 & \text{if } w_0 + \sum_{i=1}^{n} w_i x_i > 0 \\ 0 & \text{otherwise} \end{cases}$$

2b. update each weight

$$\Delta w_i = \eta (y - o) x_i$$

η is *learning rate*;

set to value << 1

$$w_i \leftarrow w_i + \Delta w_i$$

Representational power of perceptrons

perceptrons can represent only *linearly separable* concepts

$$o = \begin{cases} 1 & \text{if } w_0 + \sum_{i=1}^{n} w_i x_i > 0 \\ 0 & \text{otherwise} \end{cases}$$

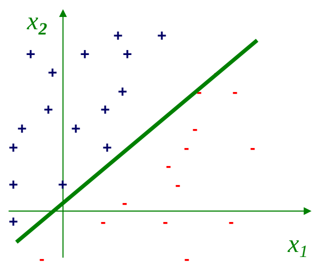
decision boundary given by: 1 if w_0

$+w_1x_1 + w_2x_2 > 0$ $w_1x_1 + w_2x_2 = -w_0$

$\overline{w_1x_1 - w_0}$

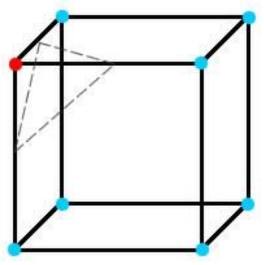
$x_2 = -$

w_2 w_2



Representational power of perceptrons

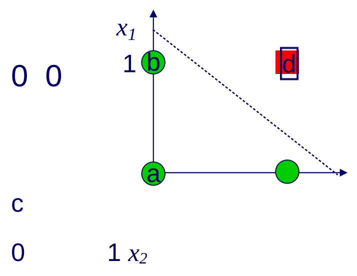
- in previous example, feature space was 2D so decision boundary was a line
- in higher dimensions, decision boundary is a hyperplane



Linearly separable functions

AND

	x_1	x_2	y
a	0	0	0
b	0	1	0
c	1	0	0
d	1	1	1

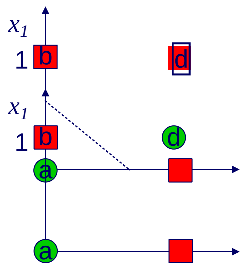


<u>OR</u>

x_1 x_2 y a 0 0 0 b 0 1 1

c 1 0 1 c

d 1 1 1 0 1 x_2



Is XOR linearly separable?

d 1 1 0 0 1 x_2

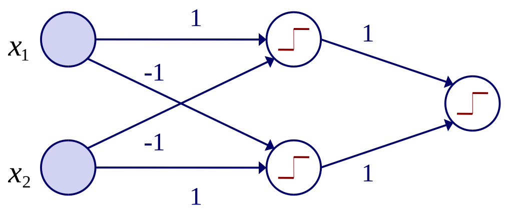
x_1 x_2 y a

0 0 0 b

0 1 1

c 1 0

c

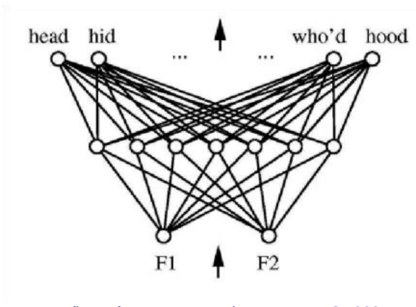


1

a multilayer perceptron
can represent XOR

assume $w_0 = 0$ for all nodes

Example of multi-layer neural network



output units

hidden units input

units

figure from Huang & Lippmann, *NIPS* 1988

input: two features from spectral analysis of a spoken sound output:

vowel sound occurring in the context “h__d”

Decision regions of a multi-layer NN

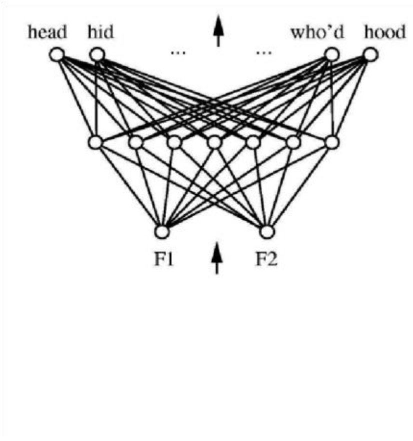
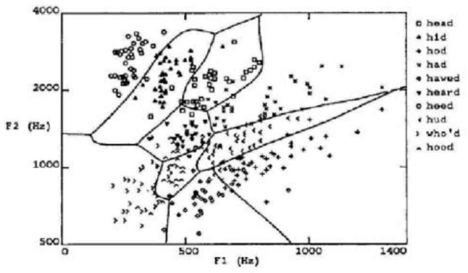


figure from Huang & Lippmann*NIPS* 1988

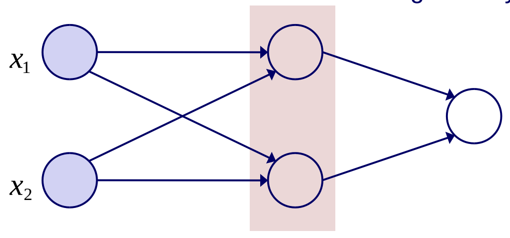


input: two features from spectral analysis of a spoken sound output:

vowel sound occurring in the context “h__d”

Learning in a multi-layer NN

- work on neural nets fizzled in the 1960's
- single layer networks had representational limitations (linear separability)
- no effective methods for training multilayer networks



how to determine error signal for *hidden units?*

- revived again with the invention of *backpropagation* method [Rumelhart & McClelland, 1986; also Werbos, 1975]
- key insight: require neural network to be differentiable; use *gradient descent*

Gradient descent in weight space

Given a training set $D = \{(\boldsymbol{x}^{(1)}, y^{(1)}) \ldots (\boldsymbol{x}^{(m)}, y^{(m)})\}$ we can specify an error measure that is a function of our weight vector $\boldsymbol{w}$

$$1 \qquad (d) \qquad (d)^{2}$$

$$E(\boldsymbol{w}) = \frac{}{2} \sum_{d \in D} \left(y - o\right)$$

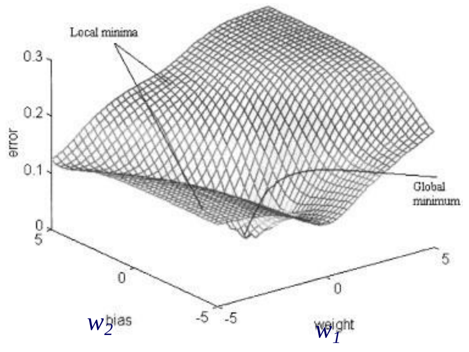


figure from Cho & Chow, *Neurocomputing* 1999

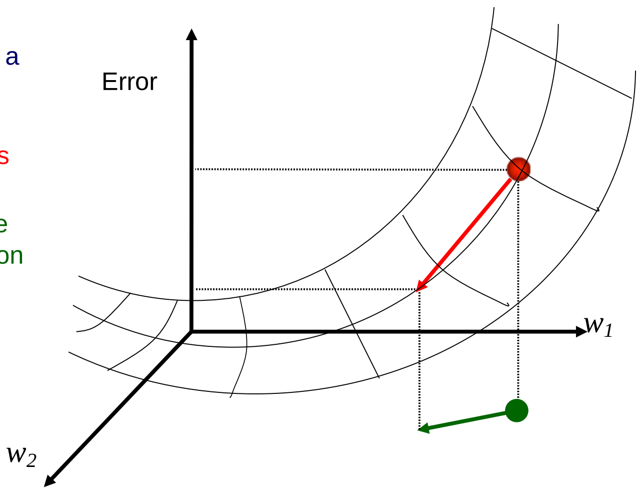
This *objective function* defines a surface over the model (i.e. weight) space

Gradient descent in weight space

gradient descent is an iterative process aimed at finding a minimum in the error surface

on each iteration

- current weights define a point in this space
- find direction in which error surface descends most steeply
- take a step (i.e. update weights) in that direction



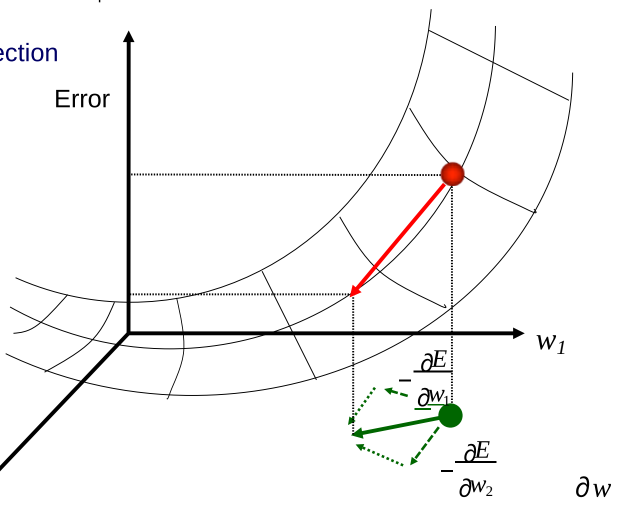
Gradient descent in weight space

calculate the gradient of E: $\nabla E(\mathbf{w}) = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n} \right]$

take a step in the opposite direction

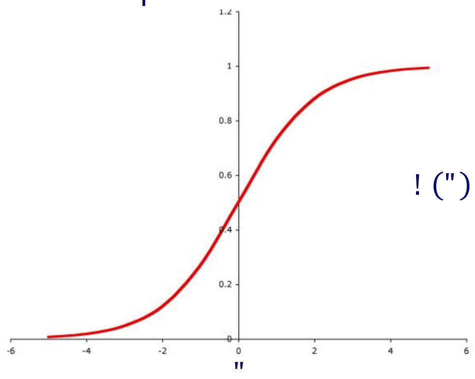
$$\Delta \mathbf{w} = -\eta \, \nabla E(\mathbf{w})$$

$$\Delta w_i = -\eta \, \frac{\partial E}{\partial w_i}$$


Error
w_1
w_2
$-\frac{\partial E}{\partial w_1}$
$-\frac{\partial E}{\partial w_2}$
∂w

The sigmoid function

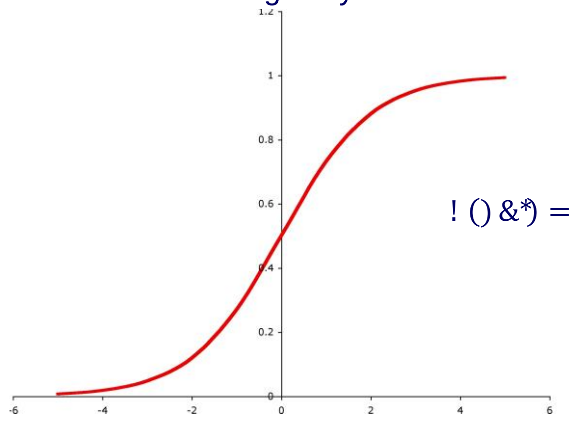
- to be able to differentiate E with respect to w_i, our network must represent a continuous function
- to do this, we can use *sigmoid functions* instead of threshold functions in our hidden and output units



$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

The sigmoid function

for the case of a single-layer network



! () &*) = $\dfrac{1}{1 + \&^{'\,(+,\ -\ \Sigma_/ +_/ 1_/)}}$

) &*= 2 $_3$ + 4 $_5$ 2 $_5$ 6 $_5$

BatchNN training

given: network structure and a training set $D=\left\{(\boldsymbol{x}^{(1)}, y^{(1)})\square(\boldsymbol{x}^{(m)}, y^{(m)})\right\}$

initialize all weights in $\boldsymbol{w}$ to small random numbers

until stopping criteria met do initialize the error

$E(\boldsymbol{w}) = 0$

for each $(\boldsymbol{x}^{(d)}, y^{(d)})$ in the training set input $\boldsymbol{x}^{(d)}$ to the network and compute output $o^{(d)}$ increment the error

$E(\boldsymbol{w}) = E(\boldsymbol{w}) + \frac{1}{2}\left(y^{(d)} - o^{(d)}\right)^2$ calculate the gradient

$$\nabla E(\boldsymbol{w}) = \Big| \#\ \frac{\partial E}{\partial}, \frac{\partial E}{\partial}, w_0\, w_1 \square, \frac{\partial E}{\partial w_n} \Big]\ (' \ \$$$

update the weights

$$\Delta \boldsymbol{w} = -\eta\ \nabla E(\boldsymbol{w})$$

Online vs. Batch learning

- Standard gradient descent (batch training): calculates error gradient for the <u>entire training set</u>, before taking a step in weight space
- *Stochastic gradient descent* (online training): calculates error gradient for a single instance (or a small set of instances, a “mini batch”), then takes a step in weight space
  - much faster convergence
  - less susceptible to local minima

OnlineNN training (Stochastic Gradient Descent)

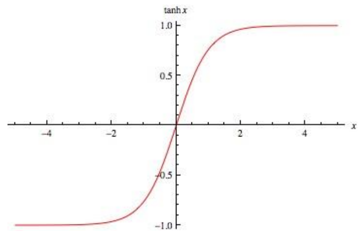
```
given: network structure and a training set $D = \{(\mathbf{x}^{(1)}, y^{(1)}) \ldots (\mathbf{x}^{(m)}, y^{(m)})\}$

initialize all weights in $\mathbf{w}$ to small random numbers until
stopping criteria met do
    for each $(\mathbf{x}^{(d)}, y^{(d)})$ in the training set input $\mathbf{x}^{(d)}$ to the
        network and compute output $o^{(d)}$ calculate the
        error $E(\mathbf{w}) = \frac{1}{2}\left(y^{(d)} - o^{(d)}\right)^2$ calculate the gradient

            $\nabla E(\mathbf{w}) = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n}\right]$

        update the weights

            $\Delta \mathbf{w} = -\eta \nabla E(\mathbf{w})$
```

Other activation functions

- the sigmoid is just one choice for an *activation function*
- there are others we can use including

hyperbolic tangent

$$!\,6\,(\ ) = \tanh 6\,(\ ) = \frac{2}{1+e^{-2x}} - 1$$



rectified linear (ReLU)

$$!\,(6\ ) \quad \begin{matrix} 0 & \text{if}\, x < 0 \\ x & \text{if} \quad x \geq 0 \end{matrix}$$

$$= >$$

Other objective functions

- squared error is just one choice for an *objective function*
- there are others we can use including cross entropy

$$E\,F(\ ) = \sum_{H\in N} -y^{(d)}\ln\left(o^{(d)}\right) - (1 - \quad (\ ))\ G^{H}\ln\left(1 - J^{(H)}\right)$$

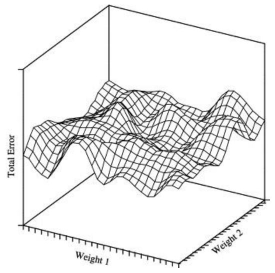
multiclass

cross entropy

$$E\,F(\ ) = -\sum_{H\in N}\ \sum_{5VW}^{\#\quad QRSTTUT} y_i^{(d)}\ O)\ \left(J5(H)\right)$$

Convergence of gradient descent

- gradient descent will converge to a minimum in the error function
- for a multi-layer network, this may be a *local minimum* (i.e. there may be a “better” solution elsewhere in weight space)



- for a single-layer network, this will be a global minimum (i.e. gradient descent will find the “best” solution)
- Recent analysis suggests that local minima are probably rare in high dimensions; saddle points are more of a challenge [Dauphin et al., NIPS 2014]